3.8 Richer Type Annotations

Recall our definition of `max_length` from the previous section:

```
def max_length(strings: set) -> int:
    """Return the maximum length of a string in the set of strings.

    Preconditions:
      - len(strings) > 0
    """
    return max({len(s) for s in strings})
```

Let us introduce another issue:

```
>>> max_length({1, 2, 3})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in <setcomp>
TypeError: object of type 'int' has no len()
```

Once again, our specification of valid inputs has failed us. The parameter type for `max_length` is `set`, and in Python `set`s can contain the values of many different types. It is not until the function description that we see that the parameter is not just *any* set, but specifically a set of strings. We could make this requirement more explicit by introducing another precondition, but there is a better approach. In this section, we'll learn how to use Python's `typing` module to increase the specificity of our type annotations.

The types in a collection

There are four collection types that have seen so far: `set`, `list`, `tuple`, and `dict`. These are analogous to the data types we've been using so far, with one key exception: we can specify the types of the values they can contained by writing them in square brackets. The table below shows these types and some examples; note that `T`, `T1`, etc. are variables that could be replaced with any data type.

Type	**Description**
`set[T]`	A set whose elements all have type `T`
`list[T]`	A list whose elements all have type `T`
`tuple[T1, T2, ...]`	A tuple whose first element has type `T1`, second element has type `T2`, etc.

Type	Description
`dict[T1, T2]`	A dictionary whose keys are of type `T1` and whose values are of type `T2`

For example:

- `{'hi', 'bye'}` has type `set[str]`
- `[1, 2, 3]` has type `list[int]`
- `('hello', True, 3.4)` has type `tuple[str, bool, float]`
- `{'a': 1, 'b': 2, 'c': 3}` has type `dict[str, int]`

Here is how we can improve the type contract for `max_length`:

```
def max_length(strings: set[str]) -> int:
    """Return the maximum length of a string in the set of strings.

    Preconditions:
      - len(strings) > 0
    """
    return max({len(s) for s in strings})
```

General collections

Though indicating the type of the values inside a collection is useful, it is not always necessary. Sometimes we *want* to be flexible and say that a value must be a list, but we don't care what's in the list (could be a list of strings, a list of integers, or a list of strings mixed with integers). Or, we might want a list (or other collection) with elements of different types. In such cases, we will continue using the built-in types `set`, `list`, `tuple`, and `dict`, for these types annotations *without* additional information.

Applying what we've learned

Let us revisit a function we designed when discussing if statements:

```
def get_status_v3(scheduled: int, estimated: int) -> str:
    """Return the flight status for the given scheduled and estimated
         departure times.

    The times are given as integers between 0 and 23 inclusive, representing
    the hour of the day.

    The status is 'On time', 'Delayed', or 'Cancelled'.
    """
```

How can we improve the specification of this function? Looking at the type annotations we see that, since none are collection types, we cannot make them any more specific than they already are. Next, looking at the docstring we see that there is the potential for some preconditions:[1]

[1] We kept the English description of what the times represent, but moved the Python-checkable part into formal preconditions.

```
def get_status_v3(scheduled: int, estimated: int) -> str:
    """Return the flight status for the given scheduled and estimated
        departure times.

    The times given represent the hour of the day.

    Preconditions:
      - 0 <= scheduled <= 23
      - 0 <= estimated <= 23
    """
```

Next let us revisit the `count_cancelled` function we designed:

```
def count_cancelled(flights: dict) -> int:
    """Return the number of cancelled flights for the given flight data.

    flights is a dictionary where each key is a flight ID,
    and whose corresponding value is a list of two numbers, where the first
        is
    the scheduled departure time and the second is the estimated departure
        time.

    >>> count_cancelled({'AC110': [10, 12], 'AC321': [12, 19], 'AC999': [1,
        1]})
    1
    """
    cancelled_flights = {id for id in flights
                         if get_status2(flights[id][0], flights[id][1]) ==
        'Cancelled'}
    return len(cancelled_flights)
```

Here we can improve the type annotations. The first parameter is not just a `dict`, but a `dict[str, list[int]]`—that is, its keys are strings (the flight IDs), and the corresponding value is a list of integers. Does this type annotation mean that now the documentation describing the dictionary is irrelevant? No: while the type annotation gives some insight on the structure of the data, it does not provide domain-specific context, like the fact that the `str` keys represent flight IDs, or that the list values represent scheduled and estimated arrival departure times.

There is one more precondition that we can formalize, though: the length of each list in our dictionary. *Every* list should have length two, which translates naturally into a use of Python's `all` function:

```
def count_cancelled(flights: dict[str, list[int]]) -> int:
    """Return the number of cancelled flights for the given flight data.

    flights is a dictionary where each key is a flight ID,
    and whose corresponding value is a list of two numbers, where the first
        is
    the scheduled departure time and the second is the estimated departure
        time.

    Precondition:
      - all(len(flights[k]) == 2 for k in flights)

    >>> count_cancelled({'AC110': [10, 12], 'AC321': [12, 19], 'AC999': [1,
        1]})
    1
    """
    cancelled_flights = {id for id in flights
                         if get_status2(flights[id][0], flights[id][1]) ==
        'Cancelled'}
    return len(cancelled_flights)
```

References

- B.4 `typing`